Carlos Andres Vergara Perico

De: Carlos Andres Vergara Perico
Enviado el: martes, 17 de septiembre de 2024 16:33
Para: Camilo Correa Valencia
Asunto: Asterix Cod primer nivel
Datos adjuntos: Stratio - Backup Codes.xlsx; columnas.txt

Marca de seguimien Seguimiento
Estado de marca: Marcado

```python
import dask.dataframe as dd
import gc
import os
import dask_geopandas as dgpd
from shapely.geometry import Point
from datetime import datetime

# Cargar el archivo GeoJSON (FIRv8) usando dask-geopandas
geojson_path = "/mnt/Asterix/Geo/FIRv8.geojson"
gdf = dgpd.read_file(geojson_path)

def punto_en_poligono(longitudes, latitudes, columna):
    """
    Función que determina si un conjunto de puntos (latitud, longitud) está dentro de
polígonos.
    Devuelve el valor de la columna especificada.

    longitudes: Columna de longitudes.
    latitudes: Columna de latitudes.
    columna: Columna geoespacial de la que se quiere obtener el valor si el punto está
dentro del polígono.
    """
    puntos = [Point(lon, lat) for lon, lat in zip(longitudes, latitudes)]
    return dgpd.sjoin_nearest(gdf, dgpd.GeoDataFrame(geometry=puntos),
how="left")[columna]

# Activar caché si se desea mejorar el rendimiento (opcional)
from dask.cache import Cache
cache = Cache(2e9)  # 2GB de caché
cache.register()

def procesar_semana(week_num, year=2024):
    """
    Función optimizada para procesar los datos de una semana específica, agregar columnas
geoespaciales y almacenar en Parquet.

    week_num: Número de la semana (entero).
    year: Año para el cual se va a procesar (por defecto 2024).
```

```
    """
    # Liberar memoria antes de comenzar
    gc.collect()

    # Identificar el primer día de la semana (lunes) basado en el número de semana y año
    first_day_of_week = datetime.strptime(f'{year} {week_num} 1', '%G %V %u')

    # Determinar el mes del primer día de la semana
    month = first_day_of_week.month

    # Definir el patrón para los archivos Parquet que corresponden al mes identificado
    file_path_pattern = f"/mnt/Asterix/year={year}/month={month}/*.parquet"

    # Limitar las columnas leídas para reducir el uso de memoria, incluyendo las columnas
adicionales
    columnas_necesarias = ['I380_02_TID', 'I245_01_TID', 'I390_02_CSN', 'I136_01_MFL',
'I135_01_TBA',
                           'FECHA_MENSAJE', 'I070_01_TOD', 'I105_01_LAT', 'I105_01_LON',
                           'I390_03_TYP', 'I390_07_DEP', 'I390_08_DST']

    # Leer los archivos del mes correspondiente, limitando las columnas para reducir el
uso de memoria
    df = dd.read_parquet(file_path_pattern, columns=columnas_necesarias)

    # Convertir 'FECHA_MENSAJE' a datetime en Dask
    df['FECHA_MENSAJE'] = dd.to_datetime(df['FECHA_MENSAJE'])

    # Convertir explícitamente 'I070_01_TOD' de 'decimal.Decimal' a 'int'
    df['I070_01_TOD'] = df['I070_01_TOD'].astype(float).fillna(0).astype(int)

    # Crear la columna de número de semana ISO
    df['Time_WeekNum'] = df['FECHA_MENSAJE'].dt.isocalendar().week

    # Filtrar los datos por la semana proporcionada
    df_filtrado = df[df['Time_WeekNum'] == week_num]

    # Liberar memoria innecesaria
    del df
    gc.collect()

    # Reparticionar adaptativamente para optimizar el uso de memoria y CPU
    df_filtrado = df_filtrado.repartition(npartitions="auto")  # Reparticionamiento
automático

    # Crear las columnas necesarias después del filtrado
    df_filtrado['Vuelo'] =
df_filtrado['I380_02_TID'].combine_first(df_filtrado['I245_01_TID']).combine_first(df_filt
rado['I390_02_CSN'])
    df_filtrado['Aerolínea'] = df_filtrado['Vuelo'].str.slice(0, 3)
    df_filtrado['Altitud'] =
df_filtrado['I136_01_MFL'].combine_first(df_filtrado['I135_01_TBA'])
```

```
    # Calcular 'fecha_hora_utc' sumando el tiempo en segundos (I070_01_TOD son los
segundos desde la medianoche)
    df_filtrado['Fecha y hora UTC'] = df_filtrado['FECHA_MENSAJE'] +
df_filtrado['I070_01_TOD'].astype('timedelta64[s]')

    # Aplicar la función de punto en polígono para FIR y Control usando operaciones
vectorizadas
    df_filtrado['FIR'] = punto_en_poligono(df_filtrado['I105_01_LON'],
df_filtrado['I105_01_LAT'], 'FIR')
    df_filtrado['Control'] = punto_en_poligono(df_filtrado['I105_01_LON'],
df_filtrado['I105_01_LAT'], 'Control')
    df_filtrado['FIRsector'] = punto_en_poligono(df_filtrado['I105_01_LON'],
df_filtrado['I105_01_LAT'], 'FIRsector')
    df_filtrado['FIRall'] = punto_en_poligono(df_filtrado['I105_01_LON'],
df_filtrado['I105_01_LAT'], 'FIRall')

    # Eliminar duplicados basados en las columnas 'FECHA_MENSAJE', 'Vuelo', 'I070_01_TOD'
    df_filtrado = df_filtrado.drop_duplicates(subset=['FECHA_MENSAJE', 'Vuelo',
'I070_01_TOD'])

    # Liberar memoria innecesaria
    gc.collect()

    # Seleccionar y organizar las columnas en el orden solicitado
    df_final = df_filtrado[['Aerolínea', 'Vuelo', 'Fecha y hora UTC',
                            'I105_01_LAT', 'I105_01_LON', 'Altitud',
                            'I390_03_TYP', 'I390_07_DEP', 'I390_08_DST',
                            'FIR', 'Control', 'FIRsector', 'FIRall']]

    # Renombrar las columnas para mayor claridad
    df_final = df_final.rename(columns={
        'I105_01_LAT': 'Latitud',
        'I105_01_LON': 'Longitud',
        'I390_03_TYP': 'Modelo del avión',
        'I390_07_DEP': 'Origen',
        'I390_08_DST': 'Destino'
    })

    # Definir la ruta de salida para guardar los archivos Parquet organizados por año,
mes, semana
    output_dir = f"/mnt/Asterix/_Temp/year={year}/month={month}/week={week_num}"

    # Crear el directorio si no existe
    os.makedirs(output_dir, exist_ok=True)

    # Guardar el archivo en formato Parquet en la ubicación especificada, particionado por
año, mes, y semana
    df_final.to_parquet(output_dir, write_index=False, partition_on=['year', 'month',
'week'])

    print(f"Archivos guardados en {output_dir}")

# Ejemplo de uso de la función
```

```
procesar_semana(23)  # Procesar la semana 23 del año 2024
```